

# Introduction to Programming

## LECTURE 2 : GDB AND ASSEMBLY

MTech CS – First Year  
Indian Statistical Institute

Sourav Sen Gupta  
sg.sourav@gmail.com

## About this course

---

### **Introduction to Programming (A1)**

Target audience : MTech (CS) First Year

Instructor : Sourav Sen Gupta (sg.sourav@gmail.com)

- Lectures : Monday and Thursday (11:15-13:00)
- Venue : CSSC Lab, 4th Floor, Library Building
- Assign : Mid-Sem : End-Sem = 20 : 30 : 50

[www.souravsengupta.com/int2pro2014/](http://www.souravsengupta.com/int2pro2014/)

## Coding environment

---

**Server : 192.168.54.156**

Login : `ssh -X mtc14{RR}@192.168.54.156`

where `mtc14{RR}` is your MTech roll number.

- Create working directory : `mkdir int2pro2014`
- Go to working directory : `cd int2pro2014`
- Create daywise directory : `mkdir lecture{MM}{DD}`
- Go to daywise directory : `cd lecture{MM}{DD}`
- Create file : `gedit mtc14{RR}-{MM}{DD}-prog{NN}.c &`

# Learn C through GDB

## What is GDB?

---

GDB : GNU debugger for several languages, including C/C++

- Interactive shell for integrated execution and debugging
- Runs a program *normally* if there are no errors
- Produces *useful* error messages if the program crashes
- Allows inspection of the program at specific points during execution
- Allows inspection of the program step-by-step during execution
- Allows inspection and tracking of parameters during execution

GDB is the *best* environment to learn C

## Using GDB

---

Write the following code and save it as 'simple.c'

```
int main() {  
    int iNum1, iNum2;  
    iNum1 = 3;  
    iNum2 = iNum1 + 10;  
    return 0;  
}
```

Compile for execution : `$ gcc -Wall simple.c -o simple`

Execute in terminal : `$ ./simple`

## Using GDB

---

Write the following code and save it as 'simple.c'

```
int main() {  
    int iNum1, iNum2;  
    iNum1 = 3;  
    iNum2 = iNum1 + 10;  
    return 0;  
}
```

Compile for debugging : `$ gcc -Wall -g simple.c -o simple`

Execute within GDB : `$ gdb simple`

## Using GDB

---

Simple execution, as in the terminal.

```
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple
[Inferior 1 (process 5325) exited normally]
(gdb)
```

## Using GDB

---

Execution with a specific *break*-point.

```
(gdb) break simple.c:2
Breakpoint 1 at 0x4004d0: file simple.c, line 2.
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple

Breakpoint 1, main () at simple.c:3
3  iNum1 = 3;
(gdb) continue
Continuing.
[Inferior 1 (process 5520) exited normally]
(gdb)
```

## Using GDB

---

Execution with multiple *break*-points.

```
(gdb) break simple.c:2
Breakpoint 1 at 0x4004d0: file simple.c, line 2.
(gdb) break simple.c:4
Breakpoint 2 at 0x4004d7: file simple.c, line 4.
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple
```

```
Breakpoint 1, main () at simple.c:3
3  iNum1 = 3;
(gdb) continue
Continuing.
```

```
Breakpoint 2, main () at simple.c:4
4  iNum2 = iNum1 + 10;
(gdb) continue
Continuing.
[Inferior 1 (process 5866) exited normally]
```

## Using GDB

---

Execution with *break*-point at specific function(s).

```
(gdb) break main
Breakpoint 1 at 0x4004d0: file simple.c, line 3.
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple

Breakpoint 1, main () at simple.c:3
3   iNum1 = 3;
(gdb) continue
Continuing.
[Inferior 1 (process 6029) exited normally]
```

## Using GDB

---

Step-by-step execution with *break-point(s)*.

```
(gdb) break main
Breakpoint 1 at 0x4004d0: file simple.c, line 3.
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple

Breakpoint 1, main () at simple.c:3
3   iNum1 = 3;
(gdb) step
4   iNum2 = iNum1 + 10;
(gdb) step
5   return 0;
(gdb) step
6 }
(gdb) continue
Continuing.
[Inferior 1 (process 6242) exited normally]
```

## Using GDB

---

Printing intermediate values during execution.

```
(gdb) break main
Breakpoint 1 at 0x4004d0: file simple.c, line 3.
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple

Breakpoint 1, main () at simple.c:3
3  iNum1 = 3;
(gdb) print iNum1
$1 = 0
(gdb) step
4  iNum2 = iNum1 + 10;
(gdb) print iNum1
$2 = 3
(gdb) step
5  return 0;
(gdb) print iNum2
$3 = 13
```

## Using GDB

---

Printing intermediate values during execution.

```
(gdb) break main
Breakpoint 1 at 0x4004d0: file simple.c, line 3.
(gdb) run
Starting program: /home/sourav/Desktop/MTechCS/codes/simple

Breakpoint 1, main () at simple.c:3
3  iNum1 = 3;
(gdb) step
4  iNum2 = iNum1 + 10;
(gdb) step
5  return 0;
(gdb) print &iNum1
$4 = (int *) 0x7fffffff35c
(gdb) print &iNum2
$5 = (int *) 0x7fffffff358
```

## Using GDB

---

One may also use GDB for

- Modify the intermediate variables
- Call functions linked into the program
- Examine the call stack in details
- Watch a variable throughout execution
- Examine the memory and processor registers

and of course, to *debug* using all its resources!

`http://www.gnu.org/software/gdb/`

`http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html`

# Learn C through Assembly

## Assembly Code

---

Write the following code and save it as 'simple.c'

```
int main() {
    int iNum1, iNum2;
    iNum1 = 3;
    iNum2 = iNum1 + 10;
    return 0;
}
```

Compile for debugging : `$ gcc -Wall -g simple.c -o simple`

Execute within GDB : `$ gdb simple`

## Assembly Code

---

Viewing the assembly code within GDB.

```
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000004004cc <+0>:    push   %rbp
0x0000000004004cd <+1>:    mov    %rsp,%rbp
0x0000000004004d0 <+4>:    movl   $0x3,-0x4(%rbp)
0x0000000004004d7 <+11>:   mov    -0x4(%rbp),%eax
0x0000000004004da <+14>:   add    $0xa,%eax
0x0000000004004dd <+17>:   mov    %eax,-0x8(%rbp)
0x0000000004004e0 <+20>:   mov    $0x0,%eax
0x0000000004004e5 <+25>:   pop    %rbp
0x0000000004004e6 <+26>:   retq
End of assembler dump.
```

Let's analyze the assembly code line-by-line.

## Assembly Code

---

Analyzing the program through its assembly code output.

```
push    %rbp
mov     %rsp,%rbp
movl   $0x3,-0x4(%rbp)
mov     -0x4(%rbp),%eax
add    $0xa,%eax
mov     %eax,-0x8(%rbp)
mov     $0x0,%eax
pop     %rbp
retq
```

Addr.	Byte	Item
(%rbp)	0	Base
-0x1(%rbp)	-1	iNum1
-0x2(%rbp)	-2	
-0x3(%rbp)	-3	
-0x4(%rbp)	-4	
-0x5(%rbp)	-5	
-0x6(%rbp)	-6	
-0x7(%rbp)	-7	
-0x8(%rbp)	-8	

Task : Figure out sizes for all types of variables in C this way!

THANK YOU  
for your kind attention

[www.souravsengupta.com/int2pro2014/](http://www.souravsengupta.com/int2pro2014/)

Email: [sg.sourav@gmail.com](mailto:sg.sourav@gmail.com) | Office: ext3251